EXPRESS MAIL NO. EK220865412US

PATENT
Attorney Docket No. 99-401

UNITED STATES PATENT APPLICATION

OF

Frederick J. ROEBER, Scott D. STAFFORD, Brian J. PALMER, Kevin J. BROTHERS, and David B. COUSINS

FOR

SYSTEM AND METHOD FOR LOGGING COMPUTER EVENT DATA IN A DISTRIBUTED SYSTEM

SYSTEM AND METHOD FOR LOGGING COMPUTER EVENT DATA IN A DISTRIBUTED SYSTEM

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

This invention was made with Government support under Contract No. F30602-97-0296, awarded by Defense Advanced Research Projects Agency. The Government has certain rights in this invention.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to methods and systems for logging computer program event data and, more particularly, to methods and systems for logging computer program event data in a distributed system.

Description of the Related Art

Software visualization tools are well known in the art for providing ways to analyze the flow of a computer software program during the testing and debugging process. These tools track the time and value of certain events as they occur while running the program. Such tools typically implement a concept known as "event logging" to monitor and record events as they take place within the monitored program.

Generally, event logging is implemented by "instrumenting" the computer software program, which includes adding code at key points in the monitored program. The added code creates a record of when particular events occur in the program as it is being run on the computer. The created record typically contains a set of entries for individual events, such as an entry for the type of event (e.g., I/O or bus operation), the start and end of the event, and an associated value of the event. The time of the event may be based on the system clock. A programmer may then analyze the record of events to determine what events took place and when.

Event logging is even more difficult and time consuming in a distributed system. In such a case, programmers must try to align in time events occurring across multiple computers. One conventional way programmers deal with this situation is by manually comparing the event logs collected from different computers. Programmers then try to time align them based on known operational sequences between the different computers. Not surprisingly, this approach yields highly inaccurate results.

Some conventional event monitoring systems have been implemented purely in software as event logging programs. Most event logging programs execute the monitored program on the same computer as the software that performs the event logging functions. Thus, the event logging program effectively shares hardware resources with the monitored program. Because the event logging

functions are fairly time consuming, they will interfere with the process flow of the monitored program, thereby making the recorded events less useful.

Hardware-based approaches often fail to have the necessary system components to support optimal event logging. For instance, many processors do not have access to a high resolution clock for determining when events occur. Furthermore, in distributed applications involving multiple computers, the event data cannot be correlated in time. Although some conventional systems may use a clock on each computer to time stamp the events, these clocks typically have no way of being coordinated or synchronized with one another. Nor do such systems provide an event collection program suited for collating coordinated event data from the different computers. Any such systems synchronizing clocks must be running on the same system bus.

Hybrid implementations of event monitoring systems are known in the prior art. Generally, these systems use event logging software running on dedicated hardware other than the system being monitored. The primary advantage of such systems is that they minimize any interference with the system running the monitored program. As described by Haban and Wybranietz, A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 16, NO. 2, pp. 197-211, FEBRUARY 1990 (referred to as "H&W"), specialized hardware running

event monitoring software for the specific monitoring application can be implemented in each node of a distributed system.

The H&W system includes a special "test and measurement processor" ("TMP") that facilitates event monitoring upon receipt of a data value issued by an instrumented program running on a target or monitored processor. The target processor running the monitored program issues data according to events occurring in the instrumented monitored program. The TMP receives the data issued by the monitored program and target processor as incoming data. Event monitoring software running on the TMP then decodes the incoming data and records the particular event.

The H&W system is limited in several respects, however. Most importantly is that the H&W system cannot log events produced by computer programs running on target processors located throughout a distributed system. The H&W system only allows collecting of events running on processors located on the same bus to which the TMP is connected.

U.S. Patent 5,682,328 ("the '328 patent") also describes an event logging system that addresses some of the limitations found in the H&W system. The '328 patent discloses a hybrid approach in the form of a computer control card configured onto the backplane containing the target processor. The control card has its own processor for time tagging events on the target processors it monitors. But, like the H&W system, the event logging system of the '328 patent cannot monitor

events of computer programs running on processors throughout a distributed system. The disclosed system can only log events on the target processors connected to the associated backplane.

Therefore, it is desired to have an event logging system that can accurately log events of computer software programs running on processors throughout a distributed system. Moreover, it is desired to have such a system that also minimizes any intrusion into the program flow of the monitored program.

SUMMARY OF THE INVENTION

Systems consistent with the present invention allow for accurate event logging in a distributed system. Systems consistent with the present invention also minimize intrusion into the program flow of the monitored program by efficiently collecting and logging the events of the monitored program.

To achieve these and other advantages, an event logging system consistent with the present invention collects events relating to a plurality of target programs. Each program runs on a respective target processor, and each target processor is located on a separate system bus. The system comprises a plurality of event collection cards, each for receiving events from a respective one of the plurality of target programs. Each event collection card and its respective target processor is installed on the same system bus. Furthermore, each event collection card includes: a time stamp clock for providing a time stamp when each event is received; an event memory for storing the received events; a sync interface unit for receiving a

sync signal from an external source; a sync control unit for synchronizing the time stamp clock to the sync signal received by the sync interface; and a collection control unit for time stamping the collected events according to the time stamp clock synchronized to the sync signal, and for storing the time stamped events in the event memory.

A further aspect of the present invention comprises an apparatus for collecting computer program events. The apparatus comprises an event collection card for receiving the computer program events. The event collection card includes: a time stamp clock for providing a time stamp indicating when each event is received; an event memory for storing the received events; a sync interface unit for receiving a sync signal; a sync control unit for synchronizing the time stamp clock to the sync signal received by the sync interface; and a collection control unit for time stamping the collected events according to the time stamp clock synchronized to the sync signal, and for storing the time stamped events in the event memory.

Both the foregoing general description and the following detailed description are exemplary and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings provide a further understanding of the invention and, together with the detailed description, explain the principles of the invention. In the drawings:

Fig. 1 is a block diagram of an event logging system consistent with the present invention;

Figs. 2 and 3 are block diagrams illustrating synchronization in an event logging system consistent with the present invention;

Figs. 4 and 5 are flow diagrams illustrating synchronization methods consistent with the present invention; and

Figs. 6A and 6B are flow diagrams illustrating event collection methods consistent with the present invention.

DETAILED DESCRIPTION

Embodiments of the present invention will now be described with reference to the accompanying drawings. Whenever possible, the same reference numbers represent the same or similar elements in the different drawings.

Overview

Systems and methods consistent with the present invention provide an event logging system that collects events from monitored programs in a distributed

system. To this end, the event logging system includes an event collection card for each computer in the distributed system. Computers in the distributed system may or may not be interconnected, other than through the functions of their associated event collection cards. Each event collection card collects events from all of the monitored programs running on a target processor in the computer corresponding to that card. Thus, all target processors located on the same computer send events to the same event collection card. In this specification, the term event refers to any software related event occurring in or generated by a monitored program, including an event occurring in or generated by a software thread of the monitored program.

The event logging system synchronizes the clocks of each event collection card with one another. In this way, each of the event collection cards can accurately time stamp events relative to one another. In addition, the event logging system also allows target processors to write events to the event collection cards using low data overhead. Finally, the event logging system also operates at high speed, allowing parallel processing of event collecting and event formatting.

System Organization

Fig. 1 is a block diagram of an event logging system 10 consistent with the present invention. As shown in Fig. 1, system 10 includes an event collection card 100, at least one target processor 200, and a host computer 300. Event collection card 100 is connected to target processor(s) 200 via a common backplane or bus

210, such as a PCI bus. Further, event collection card 100 communicates with host computer 300 over a link 310, which may include, for example, a system bus, a cable connection, a local area network, or a wide area network. Although Fig. 1 shows only one event collection card 100, an event logging system 10 consistent with the present invention preferably includes a plurality of cards synchronized together so as to collect events from target processors of separate computers.

Each event collection card 100 receives event information from a corresponding target processor(s) 200 and, after collecting and formatting the event information, sends the collected event information to host computer 300 for display to a user. The event information received by each event collection card 100 relates to events that occur while a corresponding target processor 200 is running a particular computer software program, often referred to as the monitored program.

To send the events as they occur while the program is running, the monitored program is instrumented with event logging requests, such as with calls to a macro, that initiate event collection by event collection card 100. The calls and macros are defined by a log interface library resident on target processor 200. A programmer may instrument the program and define a macro to send the event information using techniques well known in the art, such as those described in U.S. Patent 5,682,328, the subject matter of which is hereby incorporated by reference.

When each monitored program runs on an associated target processor 200, it first performs an event log initialization with event collection card 100. During

initialization, each program and thread thereof is assigned a unique address range. As target processor 200 writes events for that program, it writes the event information to the specified address range. Upon receipt of the event information, event collection card 100 decodes the lower address bits to determine an identification value for an event generated by a particular software thread in the monitored program. Event collection card 100 then stores the identification value, a time stamp value, and any other optional data included in the data write.

As shown in Fig. 1, event collection card 100 includes a bus interface 105, an event memory 110, and a collection control unit 115, each communicating over an event collection bus 160. Event collection card 100 further includes a time stamp clock 120, a sync control unit 125, and a sync interface 130. Also included is a boot memory 135, a control program memory 140, a computer processing unit (CPU) 145, and a network interface 150, each communicating over a local bus 162. Busses 160 and 162 communicate via a bus isolation unit 165 such that busses 160 and 162 may operate in parallel.

Bus interface 105 receives the event information from target processor 200 via bus 210 and forwards the event information to collection control unit 115 over event collection bus 160. Once the event information is received, collection control unit 115 decodes the lower address bits to determine the event identification value (ID). Control unit 115 then time stamps the event ID, along with any included event data, according to the time of time stamp clock 120. Time stamp clock 120 is

preferably a high resolution clock (e.g., an atomic clock) or counter having, for example, a resolution of one microsecond. Collection control unit 115 then stores the time-stamped event information in event storage memory 110. Event memory 110 is preferably a first-in-first-out (FIFO) memory. Finally, control unit 115 updates a FIFO count value indicating the number of events currently stored in event memory 110.

Sync control unit 125 and sync interface 130 manage the synchronization functions between different event collection cards 100 of event logging system 10. Sync interface 130 receives synchronization control signals, such as a sync pulse and/or a start/stop instruction, and converts the received signals for processing by sync control unit 125. If the synchronization signal is a signal associated with a time based global positioning system (GPS), then sync interface 130 preferably receives a synchronization control signal from an external GPS receiver known to those skilled in the art. However, sync interface 130 may include such a GPS receiver, or any other circuitry known in the art, for receiving and decoding external synchronization signals.

Sync control unit 125 then calibrates or synchronizes time stamp clock 120 based on the synchronization control signals. In one embodiment, sync control unit 125 also controls the operation of collection control unit 115 based upon a start or stop instruction received through sync interface 130. In another embodiment, collection control unit 115 receives the start or stop instruction from target

processor 200 or host computer 300. Furthermore, collection control unit 115 may forward to sync control unit 125 a start or stop request received from target processor 200 or host computer 300 requesting that event collecting begin or end. In systems consistent with the present invention, collection control unit 115 and sync control unit 125 are implemented using a field programmable gate array (FPGA) chip. By forming both control units on a FPGA chip, the speed of the event collecting is increased, thereby reducing the intrusiveness of event collection card 100 in the flow of the monitored computer program.

Once event information has been stored in event memory 110, CPU 145 intermittently receives the stored event information and formats the newly collected information for downloading to host computer 300. CPU 145 operates according to a control program stored in control program memory 140. Initially, this control program is stored in boot memory 135, which is preferably a non-volatile memory, such as a FLASH or PROM memory. During boot-up, the control program of boot memory 135 is downloaded into control program memory 140. This arrangement allows the control program in boot memory 135 to be easily updated to add new functionality to event collection card 100. If host computer 300 is busy, memory 140 may buffer events for later downloading to computer 300.

As described above, busses 160 and 162 are isolated from one another by bus isolation unit 165. When the two busses are isolated, each may communicate data irrespective of the operation of the other bus. In this way, event collection card

100 allows collection control unit 115 to receive events from bus interface 105 over bus 160, while, at the same time, CPU 145 may format event information over bus 162 for transmission to host computer 300. By providing two busses 160 and 162 operating in parallel, the speed of event collection card 100 is greatly increased. Collection control unit 115 controls the operation of bus isolation unit 165 to allow communication between busses 160 and 162 only when bus 160 is not actively receiving new events.

After CPU 145 has formatted the newly collected event information, CPU 145 sends the formatted event information over link 310 to host computer 300 via network interface 150. Host computer 300 preferably executes a software visualization tool well known in the art, such as Time Scan from Etnus Corporation or Visual Analyzer by Microsoft included in the Microsoft Visual Studio. The software visualization tool receives the formatted event information from each event collection card 100 in system 10 and processes the event information using a graphical user interface program to display the event information to a user.

Figs. 2 and 3 are block diagrams illustrating a plurality of event collection cards 100 synchronized together in an event logging system consistent with the present invention. Each arrangement allows event collection cards 100 to time synchronize their respective time stamp clocks 120 together. The event logging system can then monitor computer programs across multiple computers while accurately time stamping the events of each program relative to one another.

As shown in Fig. 2, event collection cards 100a, 100b, and 100c are connected by lines 102 in a daisy chain fashion. Each collection card 100 is connected to a corresponding target processor(s) 200a, 200b, and 200c via a respective bus 210a, 210b, and 210c. When connected in this way, the time stamp clock 120 of one of the event collection cards acts as a master synchronization clock that synchronizes clocks 120 of the other event collection cards. The particular event collection card that synchronizes the other cards to its own clock is referred to as the master card. The other event collection cards are referred to as slave cards since they synchronize to the clock of the master card. In the embodiment of Fig. 2, event collection card 100a is the master card, while event collection cards 100b and 100c are slave cards.

As described in greater detail below with respect to Fig. 4, master card 100a sends a sync signal over line 102a to slave card 100b. Slave card 100b then synchronizes its time stamp clock based on the sync signal received over line 102a. Further, slave card 100b forwards the sync signal to slave card 100c over line 102b. Like slave card 100b, slave card 100c will then synchronize its time stamp clock 120 based on the received sync signal.

Fig. 3 shows an alternative embodiment in which a plurality of event collection cards 100 are synchronized using a sync signal received from an external source. As shown in Fig. 3, the event logging system includes event collection cards 100x, 100y, and 100z connected to a corresponding target processor(s) 200x,

200y, and 200z via a respective bus 210x, 210y, and 210z. Collection cards 100 are connected to receiver circuitry 103x, 103y, and 103z, respectively, for receiving a sync signal from an external source 105. In systems consistent with the invention, external source 105 may be any device or system capable of transmitting a radio frequency high-resolution sync signal, such as a sync pulse received through a time-based GPS.

Event collection cards 100x, 100y, and 100z each receive the transmitted sync signal via receivers 103x, 103y, and 103z, respectively, and synchronize their clocks based on the received signal. Because the event collection cards 100 of Fig. 3 are not physically connected, they may each be located in distant locations. Event collection cards 100x, 100y, and 100z may then communicate with host computer 300 via a network 305, such as a wide area network (WAN). In this way, an event logging system consistent with the invention may log event information from target processors 200 distributed throughout a wide geographic area.

Figs. 2 and 3 further show an exemplary display on host computer 300 shown to a user through the event visualization software. The event visualization software resident on host computer 300 receives the formatted event information provided by event collection card 100 and displays this information to the user. As shown in Figs. 2 and 3, the event visualization software displays for each trace (i.e., a computer program, or thread thereof, running on a particular target processor) the time a particular event occurred and each event's relation in time to other events

occurring on other traces. Moreover, because the clocks of each event collection card 100 are synchronized to one another, the events of each trace are accurately time stamped in relation to one another. Thus, the display of each trace begin at a common point in time (e.g., t = 0 sec.).

Figs. 2 and 3 show traces 1 to 3 having events A to E. Each trace may be based on events collected from a different event collection card 100. As shown in the Figures, a user can accurately determine the time of each event in relation to other events. Because each trace originates at the same point in time (e.g., t = 0), the events of different traces can be accurately compared to one another. For example, the display shows to a user that event C occurred after event A and before event B.

System Operation

As described above, event logging systems consistent with the present invention preferably include a plurality of event collection cards 100 synchronized together. Figs. 4 and 5 are flow diagrams illustrating synchronization methods consistent with the present invention. In particular, each flow chart illustrates the method invoked by each event collection card 100 during the synchronization process. Fig. 4 illustrates a synchronization method for an event logging system configured as shown in Fig. 2, while Fig. 5 illustrates a synchronization method for an event logging system configured as shown in Fig. 3.

As shown in Fig. 4, the method begins with collection control unit 115 determining whether it has received a start request from target processor 200 or host computer 300 (step 405). Target processor 200 may transmit a start request when the instrumented computer software program running on target processor 200 calls a macro that requests the start of event collecting. Additionally, a user of host computer 300 may transmit a start request. The start request preferably requests that all collection cards 100 begin collecting events at the same time. However, systems consistent with the present invention may include start requests that request that only a subset of event collection cards 100 begin collecting events.

If collection control unit 115 does receive a start request, then collection control unit 115 determines whether event collection card 100 is the master card or a slave card (step 410). Only the master card can initiate the synchronization process since all other cards synchronize to the master card's clock. Thus, if event collection card 100 is a slave card, collection control unit 115 instructs sync control unit 125 to forward the request to the master card (step 415). If the neighboring card is not the master card, then the slave card forwards the request to the first upstream slave card in the daisy-chain connection, which then similarly forwards the request until it reaches the master card.

If, or once, the master card itself has received the start request, then it sends a start instruction and periodically sends a sync signal to each of the slave cards (step 420). With regard to the start instruction, it instructs event collection cards

100, identified according to the start request, to begin collecting events from target processors 200. With regard to the sync signal, it is propagated along the daisy chain to each of the slave cards in the manner described above with respect to Fig. 2

In systems consistent with the invention, the sync signal is a pulse informing sync control unit 125 that time stamp clock 120 must be at a preset time. Upon receiving the sync signal, sync control unit 125 controls time stamp clock 120 to insure that clock 120 is at the preset time. Thus, upon receipt of the sync signal, sync control unit 125 may increment time stamp clock 120 to the desired time if clock 120 has not yet reached the preset time. If, on the other hand, clock 120 reaches the preset time before the sync signal is received, then sync control unit 125 will stop time stamp clock 120 until the sync signal is received. If clock 120 reaches the preset time at the same time the sync signal is received, then clock 120 will simply continue without any interruption. The preset time is preferably hard-wired or software coded into sync control unit 115.

As described above, the master card periodically sends the sync signal to the slave cards. In systems consistent with the invention, the sync signal is sent every 512 microseconds. Thus, each time a sync signal is received, sync control unit 125 will insure that the time of clock 120 is at the appropriate multiple of 512 microseconds, as described above. In this way, all of the slave cards periodically resynchronize upon receipt of the sync signal.

Event collection card 100 then collects event information (step 425) until it receives a stop request (step 430). Although the stop request preferably requests that all event collection cards 100 stop collecting events, the stop request may request that only a subset of cards 100 stop. Further, like the start request, the stop request may be received from either target processor 200 or from host computer 300. Once a stop request has been received, sync control unit 125 determines whether it is a master or slave card (step 435). If it is a slave card, then sync control unit 125 forwards the stop request on to the master card (step 440). The stop request is forwarded to the master card in the same way the start request was forwarded, as described above with respect to step 415. Once the master card receives the stop request, it then sends a stop instruction to all slaves, effectively disabling event collection (step 445).

Fig. 5 is similar to Fig. 4, but illustrates a flow diagram for a synchronization process in which there are no master or slave cards. As described above with respect to Fig. 3, all cards in this embodiment receive a synchronization signal from the same external source 105, such as a time-based GPS. As shown in Fig. 5, the method begins with collection control unit 115 determining whether it has received a start instruction from target processor 200 or host computer 300 (step 505). Target processor 200 may transmit a start instruction when the instrumented computer software program running on target processor 200 calls a

macro that requests the start of event collecting. Additionally, a user of host computer 300 may transmit a start instruction.

The start instruction preferably identifies the absolute time at which event collection is to begin. Thus, event collecting will begin once time stamp clock 120 reaches the identified time. To synchronize all cards 100 to the same time, sync control unit 125 then receives a periodic sync signal from external source 105 (step 510).

As described above with respect to Fig. 4, the sync signal is a pulse informing sync control unit 125 that time stamp clock 120 must be at a preset time. Upon receiving the sync signal, sync control unit 125 controls time stamp clock 120 to insure that clock 120 is at the preset time. Thus, upon receipt of the sync signal, sync control unit 125 may increment time stamp clock 120 to the desired time if clock 120 has not yet reached the preset time. If, on the other hand, clock 120 reaches the preset time before the sync signal is received, then sync control unit 125 will stop time stamp clock 120 until the sync signal is received. If clock 120 reaches the preset time at the same time the sync signal is received, then clock 120 will simply continue without any interruption. The preset time is preferably hard-wired or software coded into sync control unit 115.

External source 105 periodically sends the sync signal to all event collection cards 100. In systems consistent with the invention, the sync signal is sent every 100 microseconds. Thus, each time a sync signal is received, sync control unit 125

will insure that the time of clock 120 is at the appropriate multiple of 100 microseconds, as described above. In this way, all event collection cards 100 periodically resynchronize upon receipt of the sync signal.

Once time stamp clock 120 reaches the time identified by the start instruction, then sync control unit 125 will instruct collection control unit 115 to begin collecting events (step 515). Event collection card 100 then collects event information until it receives a stop instruction (step 520). Like the start instruction, the stop instruction may be received from either target processor 200 or from host computer 300. The stop instruction preferably identifies the absolute time that event collection is to stop. Once time stamp clock 120 reaches the time identified by the stop instruction, then sync control unit 125 will instruct collection control unit 115 to stop collecting events (step 525).

Figs. 6A and 6B are flow diagrams illustrating event collection methods consistent with the present invention. In particular, Fig. 6A illustrates the event collection method for receiving event information from target processor 200. Fig. 6B, on the other hand, illustrates the processing of event information by CPU 145 prior to sending the event information to host computer 300.

As shown in Fig. 6A, target processor 200 first initializes with event collection card 100 (step 605). As described above, each monitored program along with each program thread is assigned a unique address range. As target processor 200 writes events for that program, it writes the event information to the specified

address range. Event collection card 100 then waits until it receives event information from target processor 200 (step 610). Further, if collection control unit 115 receives a stop instruction from either the master card, target processor 200, or host unit 300, then event collection card 100 stops all event collection processing (step 615).

When event information is received, collection control unit 115 decodes the lower address bits to determine the event ID value of the monitored computer program corresponding to the program thread that sent the event. Collection control unit 115 then time tags the ID value and any other data included in the event write, based on the current time of time stamp clock 120, and stores this information in event memory 110 (step 620). At about the same time, collection control unit 115 also updates a FIFO event count stored in control unit 115 (step 625). The event count is monitored by CPU 145 in determining when to process the event information before sending it to host computer 300, as described below with respect to Fig. 6B.

As shown in Fig. 6B, CPU 145 reads the event count located in collection control unit 115 to determine if memory 110 contains any new events (steps 630 and 635). As described above, the event count identifies the number of new events stored in event memory 110. Preferably, the event count is a total of all events currently stored in event memory 110. Collection control unit 115 increments the event count each time an event is stored and decrements the event count each time

an event is read by CPU 145. Thus, CPU 145 may read the event count to determine if any new events are stored in event memory 110 and waiting to be processed by CPU 145.

CPU 145 preferably reads the event count during periods of inactivity on event collection bus 160. In this way, CPU 145 limits any intrusion into the collection of events from target processor 200. CPU 145 may also read the event count, however, on a periodic basis or upon the receipt of an event count interrupt signifying that event memory 110 has reached a predetermined storage threshold. Furthermore, for CPU 145 to read the event count, bus isolation unit 165 first allows access between busses 160 and 162.

To this end, CPU 145 sends a request to collection control unit 115 requesting that control unit 115 forward to CPU 145 any new events stored in event memory 110. Collection control unit 115 instructs bus isolation unit 165 to grant CPU 145 access to event collection bus 160 during periods of inactivity. During these periods, CPU 145 may communicate with and/or access event memory 110 and collection control unit 115.

If event memory 110 has not stored any new events, then, unless CPU 145 receives a stop instruction, CPU 145 will wait to a later time to again read the event count (steps 640 and 645). The stop instruction is the same as that described above with respect to step 615. If new events are stored in memory 110, then CPU 145 downloads the event information into memory (e.g., memory 140) for formatting

into a format compatible with the event visualization software resident on host computer 300 (step 650). Once the event information is reformatted, CPU 145 sends the formatted event information to host computer 300 (step 655), and processing returns to step 640. If link 310 is busy, CPU 145 may store the formatted events in memory until after event collection is completed.

Conclusion

Systems and methods consistent with the present invention collect events from computer software programs running on multiple computers throughout a distributed system. Such systems and methods consistent with the present invention can monitor any program for execution in a multiprocessor system regardless of the computer programming language. For example, both C++ and Java are programming languages commonly used to develop programs for execution by multiprocessor computer systems.

The foregoing description of preferred embodiments of the present invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications and variations are possible in light of the above teachings or may be acquired from practice of the invention. For example, the described implementation includes software and hardware, but elements of the present invention may be implemented as a combination of hardware and software, in software alone, or in hardware alone.

Further, the invention may be implemented with both object-oriented and non-object-oriented programming systems.

Although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, CD-ROM, an Internet connection, or other forms of RAM or ROM. The scope of the invention is defined by the claims and their equivalents.